A Library of Optimization Algorithms for Organizational Design

Georgiy M. Levchuk
Yuri N. Levchuk
Jie Luo
Fang Tu
Krishna R. Pattipati

Dept. of Electrical and Computer Engineering
University of Connecticut
Storrs, CT 06269-2157
Tel./Fax: (860) 486-2890/5585
E-mail: Krishna@engr.uconn.edu

Abstract

This paper presents a library of algorithms to solve a broad range of optimization problems arising in the normative design of organizations to execute a specific mission. The use of specific optimization algorithms for different phases of the design process leads to an efficient matching between the mission structure and that of an organization and its resources/constraints.
This library of algorithms forms the core of our design software environment for synthesizing organizations that are congruent with their missions. It allows an analyst to obtain an acceptable trade-off among multiple objectives and constraints, as well as between computational complexity and solution efficiency (desired degree of sub-optimality).

1. Introduction

1.1 *Motivation*

The optimal organizational design problem is one of finding both the optimal organizational structure (e.g., decision hierarchy, allocation of resources and functions to decision-makers (DMs), communication structure, etc.) and strategy (allocation of tasks to DMs, sequence of task execution, etc.) that allow the organization to achieve superior performance while conducting a specific mission ([Levchuk *et al.*, 1999a]). Over the years, research in organizational decision-making has demonstrated that there exists a strong functional dependency between the specific structure of a mission environment and the concomitant optimal organizational design. Subsequently, it has been concluded that the optimality of an organizational design ultimately depends on the actual mission parameters (and organizational constraints). This premise led to the application of systems engineering techniques to the design of human teams. This approach advocates the use of normative algorithms for optimizing human team performance (e.g., [Pete *et al.*, 1993, 1998], [Levchuk *et al.*, 1996, 1997, 1999a,b]).

* This work was supported by the Office of Naval Research under contracts #N00014-93-1-0793, #N00014-98-1-0465 and #N00014-00-1-0101

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
2005		**00-00-2005 to 00-00-2005**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
A Library of Optimization Algorithms for Organizational Design	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Office of Naval Research,One Liberty Center,875 North Randolph Street Suite 1425,Arlington,VA,22203-1995**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES
The original document contains color images.

14. ABSTRACT
This paper presents a library of algorithms to solve a broad range of optimization problems arising in the normative design of organizations to execute a specific mission. The use of specific optimization algorithms for different phases of the design process leads to an efficient matching between the mission structure and that of an organization and its resources/constraints. This library of algorithms forms the core of our design software environment for synthesizing organizations that are congruent with their missions. It allows an analyst to obtain an acceptable trade-off among multiple objectives and constraints, as well as between computational complexity and solution efficiency (desired degree of sub-optimality).

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			
unclassified	**unclassified**	**unclassified**		**40**	

1.2 ***Related Research***

When modeling a complex mission and designing the corresponding organization, the variety of mission dimensions (e.g., functional, geographical, terrain), together with the required depth of model granularity, determine the complexity of the design process. Our mission modeling and organizational design methodology allow one to overcome the computational complexity by synthesizing an organizational structure via an iterative solution of a sequence of smaller and well-defined optimization problems ([Levchuk *et al.*, 1997]). The above methodology was used to specify an organizational design software environment, outlined in [Levchuk *et al.*, 1999b], to assist a user in representing complex missions and synthesizing the organizations. The component structure of our software environment allows an analyst to mix and match different optimization algorithms at different stages of the design process.

Our mission modeling and a three-phase iterative organizational design process, first proposed in [Levchuk *et al.*, 1997] and later enhanced in [Levchuk *et al.*, 1998], is graphically represented in Figure 1.

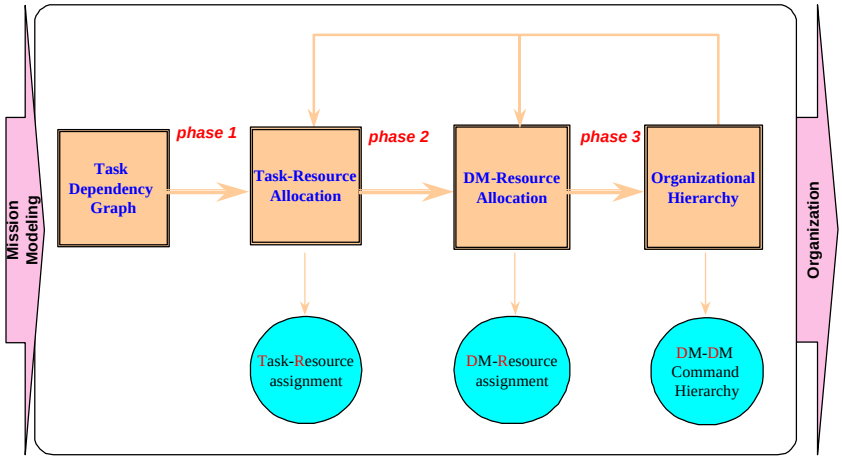

Figure 1. The 3-phase Organizational Design Process

The 3-phase design process of Figure 1 solves three distinct optimization sub-problems:

Phase I. Scheduling Phase.

In this phase, an optimal *task-resource allocation* is established. It is defined in terms of a platform-to-task assignment matrix. The objective function (mission completion time **or** a combined objective function assembled from individual mission objectives such as the completion time, accuracy, workload, expended resources, external coordination, etc.) is minimized subject to assignment, resource availability, platform velocity and graph-related (such as precedence and synchronization) constraints.

Phase II. Clustering Phase.
In this phase, an optimal *DM-resource allocation* is determined. It is referred to as DM-platform assignment matrix. The objective function (weighted sum of the maximum internal and external workloads **or** a combined objective function constructed from individual mission objectives such as the number of decision-makers, their expertise, available platforms and their resident resources, etc.) is minimized subject to assignment and DM workload constraints.

Phase III. Structural Optimization Phase.
In this phase, an optimal *organizational hierarchy* is found. It is represented in the form of a directed tree with directed arcs specifying supported-supporting relations. The objective function (maximal hierarchy workload induced by direct (one-to-one) and indirect coordination **or** a combined objective function gleaned from the identified mission objectives such as the number of communication links available for each DM, depth of organizational hierarchy, information flow, etc.) is minimized subject to the graph-related (information access and hierarchy structure) constraints.

On-line Adaptation Phase. In case of an asset or a decision node failure, the application of a branch-and-bound method to the task-resource allocation-preference matrix generates the next best assignments (the *new task-resource allocation strategy*). This method provides a quick and efficient search for adaptation options. The dynamic scheduling accounts for on-line changes without having to completely resolve the problem. If the newly obtained task-resource assignment matrix violates the organizational constraints, *Phases II* and *III* of the algorithm are used to generate the *new organizational structure*. In this case, *Phase II* is completed in an *evolutionary mode* (platform clusters are obtained by regrouping the old platform groups, rather than generating entirely new ones from scratch). Finally, if the process of generating a feasible organizational structure fails, the mission must be *aborted* (see [Levchuk *et al.*, 98] for details).

1.3 ***Scope and Organization of Paper***

In section 2, we provide an overview of our mission modeling and organizational design environment. In section 3, the optimization algorithms associated with the three optimization stages are described. Section 4 concludes with a summary and plans for future research.

2. **Multi-objective Optimization and Organizational Design Software Environment**

Recently, we have begun the process of implementing our modeling and design methodology in *software* to fully automate the organizational design process, while allowing for iterative user-defined modifications at various stages of the design process. To assist an analyst, our software environment is designed to display the metrics of organizational performance, characterize the attainment of mission objectives, and specify the workload distribution across the organizational elements of interest.

Our modeling and design environment ([Shlapak *et al.*, 2000]) includes the following seven key components (Fig.2):

(1) *Asset/Resource Description*;
(2) *DM Structure Profiler;*
(3) *Mission Modeling;*
(4) *Performance Criteria/Measures;*
(5) *Schedule Generation;*
(6) *Resource Allocation;* and
(7) *Hierarchy Construction.*

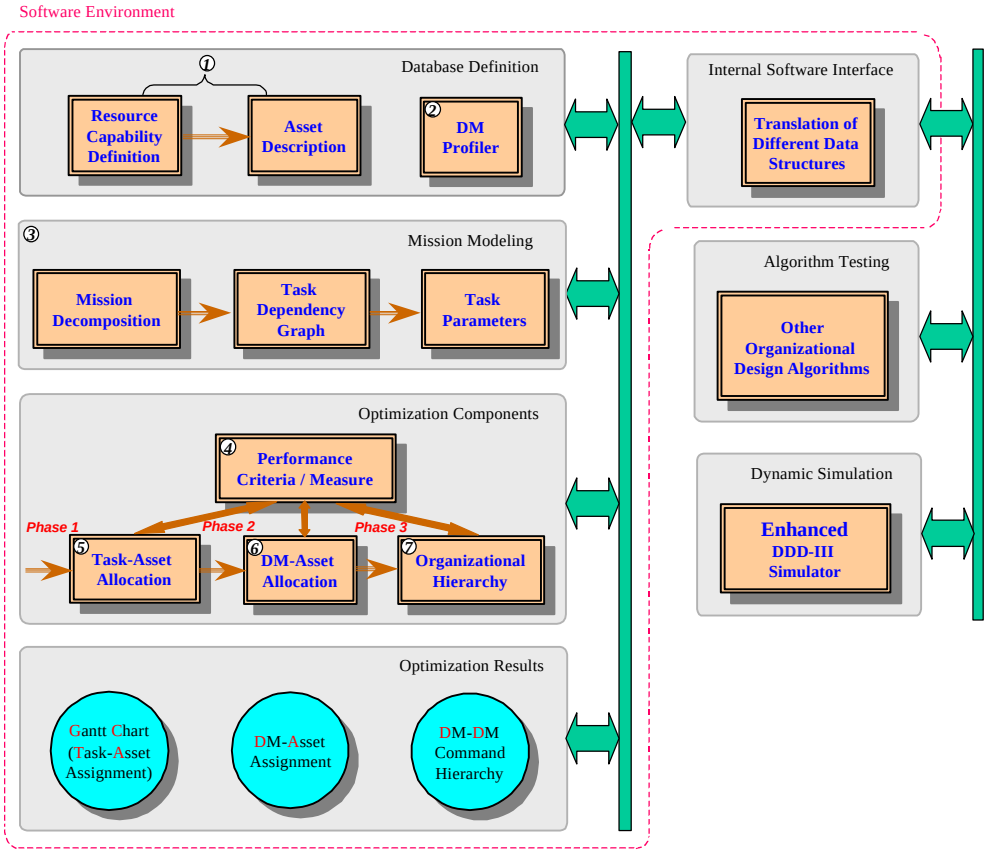

Figure 2. Component Architecture of Our Software Environment

2.1 ***Modeling Components***

The first three components of our design environment (*Asset/Resource Description*, *DM Structure Profiler*, and *Mission Modeling*) are devised to assist an analyst in developing mission models (of various complexity) and organizational constraints. These serve as inputs to our design process. The *Performance Criteria/Measures* component is used to stipulate objective functions for the design process and to define a cost function that combines mission objectives and design parameters.

2.2 ***Design Optimization Components and Algorithms***

After the *Performance Criteria/Measures* component is used to define objective functions for the design process, the last three components of our software environment (*Schedule Generation*, *Resource Allocation*, and *Hierarchy Construction*) allow an analyst to perform a step-by-step design of the organizational structure, while implementing, if desired, the user-defined design modifications at various stages of the design process to adjust the metrics of organizational performance (e.g., weights on objective function, workload distribution, etc.). These design optimization components present a step-by-step visualization of our organizational design process. Specifically, the *Schedule Generation* component produces the task-resource allocation schedule that corresponds to Phase I of our organizational design algorithm. The *Resource Allocation* component (Phase II) defines DM functionality by grouping platforms and provides a balance between internal and external coordination. Finally, the *Hierarchy Construction* component (Phase III) derives organizational hierarchy to minimize the workload due to indirect external coordination induced by the hierarchy structure.

In general, the three sub-problems of Schedule Generation, Resource Allocation, and Hierarchy Construction are NP hard (optimal algorithms take exponential time). Thus, efficient (near-optimal) heuristics need to be explored to effectively solve large-scale organizational design problems. The modular structure of our software environment allows one to apply different algorithms (both optimal and heuristic) at different stages of the design process to handle the complexity of a specific problem at hand. The iterative application of the corresponding algorithms allows us to simultaneously optimize multiple performance criteria, subject to an acceptable trade-off among design objectives.

The organizational structure, an outcome of the design process, prescribes the relationships among the organizational entities by specifying:

- Task-resource schedule;
- DM-resource access/allocation;
- DM organizational hierarchy;
- Inter-DM coordination structure.

The organizational structure defines each individual DM's *capabilities* (by assigning each DM a share of the information and resources) and specifies the rules that regulate inter-DM *coordination*. The organizational structure, together with a set of thresholds constraining a DM workload, determines the boundaries of the space of feasible organizational strategies (i.e., all feasible DM-task-resource assignments and coordination strategies), from which the organization can choose a particular strategy for implementation. The feasible strategy space delimits the strategy adaptations that an organization can undertake without having to undertake major structural reconfigurations.

3. **Library of Optimization Algorithms**

In this section, we present the library of optimization algorithms used in our organizational design software environment. The library is constantly evolving and new algorithms and performance measures are being added to enlarge the scope of applicability of our software environment.

3.1 ***Scheduling***

3.1.1 ***Problem Definition***

Scheduling concerns the allocation of limited resources to tasks over time. The resources and tasks may take many forms. The resources may be platforms, human teams, surveillance assets, information sources, etc. The tasks may be landings or take-offs, evaluations or executions, operational or informational. They can be aggregated or independent, defensive or offensive. Each task may have a different priority level and opportunity window.

Scheduling is a decision-making process that has as its goal to optimize one or more objectives. The objectives may take many forms. One possible objective is the minimization of mission completion time, and another is task deadline violation.

The scheduling phase of the organizational design process can be generally described as follows. A set of tasks with specified processing times, resource requirements, locations and precedence relations among them need to be executed by a given set of platforms with specified resource capabilities, ranges of operation and velocities. Resource requirements and resource capabilities are represented via vectors of the same length with each entry corresponding to a particular resource type. Tasks are assigned to groups of platforms in such a way that, for each such assignment, the vector of task's resource requirements is component-wise less than or equal to the aggregated resource capability of the group of platforms assigned to it. The task can begin to be processed only when all its predecessors are completed and all platforms from the group assigned to it arrive at its location. A resource can process only one task at a time. Platforms are to be routed among the tasks so that the overall completion time (called *Mission Completion Time* –the completion time of the last task) is minimized.

3.1.2 ***Example***

A joint group of Navy and Marine Forces is assigned to complete a military mission that includes capturing a seaport and airport to allow for the introduction of follow-on forces. There are two suitable landing beaches designated "North" and "South", with a road leading from the North Beach to the seaport, and another road leading from the South Beach to the airport. From intelligence sources, the approximate concentration of the hostile forces is known, and counter-strikes are anticipated. The commander devises a plan for the mission that includes the completion of tasks shown in Figure 3. The following 8 resource requirements/capabilities are modeled: AAW (Anti-Air Warfare), ASUW (Anti-Submarine Warfare), ASW (Anti-Sea Warfare), GASLT (Ground Assault), FIRE (Firing Squad), ARM (Armory), MINE (Mine Clearing), DES

(Destroyer). In Figure 4, mission tasks, the assets (platforms) available for operation, resource requirement vector for each task, resource capability vector for each platform and other relevant parameters are presented.

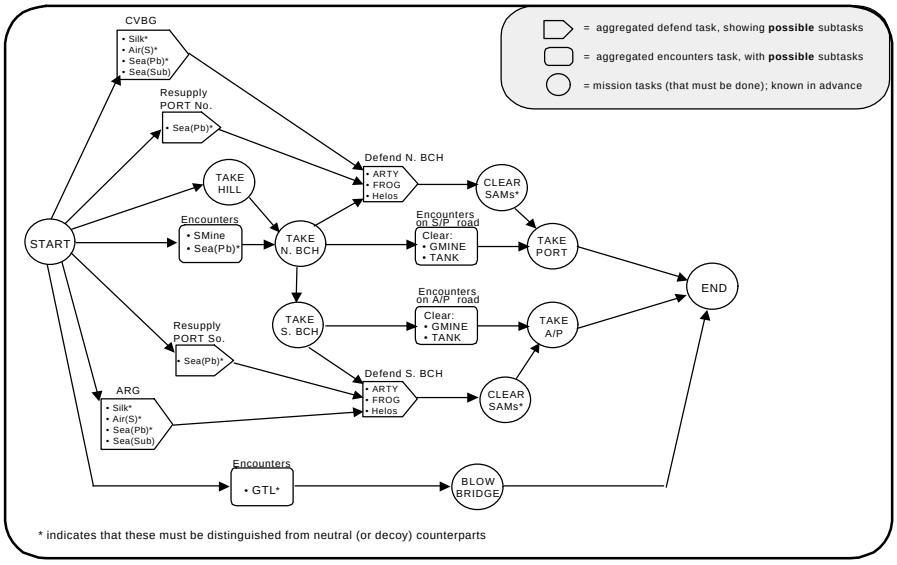

Figure 3. Task-precedence graph for Example 1.

Tasks	Locations	Resource Requirement Vector	Processing Time
1 CVBG	70 15	5 3 10 0 0 8 0 6	30
2 ARG	64 75	5 3 10 0 0 8 0 6	30
3 Resupply Port N	15 40	0 3 0 0 0 0 0 0	10
4 Resupply Port S	30 95	0 3 0 0 0 0 0 0	10
5 Encounters N&S	28 73	0 3 0 0 0 0 10 0	10
6 HILL	24 60	0 0 0 10 14 12 0 0	10
7 NORTH BEACH	28 73	0 0 0 10 14 12 0 0	10
8 SOUTH BEACH	28 83	0 0 0 10 14 12 0 0	10
9 Defend N. Beach	28 73	5 0 0 0 0 5 0 0	10
10 Defend S. Beach	28 83	5 0 0 0 0 5 0 0	10
11 S/P Road	25 45	0 0 0 0 0 10 5 0	10
12 A/P Road	5 95	0 0 0 0 0 10 5 0	10
13 SAM SeaPort	25 45	0 0 0 0 0 8 0 6	20
14 SAM AirPort	5 95	0 0 0 0 0 8 0 6	20
15 SEAPORT	25 45	0 0 0 20 10 4 0 0	15
16 AIRPORT	5 95	0 0 0 20 10 4 0 0	15
17 GTL	5 60	0 0 0 0 0 8 0 4	10
18 Blow Bridge	5 60	0 0 0 8 6 0 4 10	20

Platforms	Resource Capability Vector	Velocity
1 DDG	10 10 1 0 9 5 0 0	2
2 FFG	1 4 10 0 4 3 0 0	2
3 CG	10 10 1 0 9 2 0 0	2
4 ENG	0 0 0 2 0 0 5 0	4
5 INFA	1 0 0 10 2 2 1 0	1.35
6 SD	5 0 0 0 0 0 0 0	4
7 AH1	3 4 0 0 6 10 1 0	4
8 CAS1	1 3 0 0 10 8 1 0	4
9 CAS2	1 3 0 0 10 8 1 0	4
10 CAS3	1 3 0 0 10 8 1 0	4
11 VF1	6 1 0 0 1 1 0 0	4.5
12 VF2	6 1 0 0 1 1 0 0	4.5
13 VF3	6 1 0 0 1 1 0 0	4.5
14 SMC	0 0 0 0 0 0 10 0	2
15 TARP	0 0 0 0 0 0 0 6	5
16 SAT	0 0 0 0 0 0 0 6	7
17 SOF	0 0 0 6 6 0 1 10	2.5
18 INF (AAAV – 1)	1 0 0 10 2 2 1 0	1.35
19 INF (AAAV – 2)	1 0 0 10 2 2 1 0	1.35
20 INF (MV22 – 1)	1 0 0 10 2 2 1 0	1.35

Figure 4. Task Requirement and Platform Capability Data for Example 1

3.1.3 *Related Research*

The scheduling problem arising in organizational design extends to a large set of well-known problems. When there exists only one platform, it is related to the Traveling Salesman Problem (TSP) and its extensions (such as Time-dependent TSP, TSP with precedence relations, etc. –for review, see [Lawler *et al.*, 1985], for latest results, see [Mingozzi *et al.*, 1997], [Zweig *et al.*, 1995], [Fischetti *et al.*, 1997], [Franca, 1995]). When any platform can process any task, the problem simplifies to Multiple TSP with precedence relations. If, in addition, the processing of a task can be separated in time among different platforms, our problem is related to the Vehicle Routing problem and its extensions (for review, see [Malandraki *et al.*,1992], [Golden *et al.*, 1988], for latest results, see [Fisher *et al.*, 1994], [Dumas *et al.*, 1995], [Taillard *et al.*, 1997]).

Another related useful problem is the Dial-a-Ride problem (see [Madsen *et al.*, 1995]). In the case when travel times among task locations are much smaller than the task processing times (and therefore can be ignored), the problem reduces to a Multiprocessor Scheduling problem with Precedence Constraints (for review, see [El-Rewini, 1994], [Cheng *et al.*, 1990], for recent studies see [Chan, 1998], [Van De Velde, 1993], [Baruah, 1998]). For a review of general scheduling problems, see [Pinedo, 1995], [El-Rewini, 1994].

Other variations of problem formulation are possible. For example, there may exist a delay between processing of two tasks on the same platform ("adjustment" delay). The opposite of this situation is when the delay occurs only when tasks are processed on different platforms (communication delays) with no delay for processing by the same platform. This has relevance in Multiprocessor Scheduling with inter-processor communication delays (see [Baruah, 1998], [Selvakumar, 1994]). Another variation is the existence of time windows for processing each task (that is, the earliest start times, called *release times*, and the latest end-times, called *deadlines*, define opportunity windows for tasks). In this case, the objective function involves the minimization of earliness-tardiness penalties (that is, the penalties resulting from processing tasks outside of their time-windows). In our problem, we assume that task-processing times are fixed. In real life, situations may arise when task-processing times depend on the amount of resources allocated to them. The objective then is to achieve a tradeoff between processing tasks as fast as possible and using as little resources as possible [Turek *et al.*, 1992]. Another complication is that a task can begin to be processed when the assigned platforms are within a specified distance of this task (depending on the task and the range of the platform). In this case, the problem assumes the form of the shortest covering path problem (see [Current, 1994]). Other realistic constraints, such as the ability of tasks to move during the mission and platforms having expendable resources (such as fuel, firepower, supplies, etc.), can be included.

All of these instances of our scheduling problem are proven to be NP-hard, meaning that no known polynomial algorithms exist for finding their optimal solutions. Therefore, research in this area has primarily focused on the development of near-optimal algorithms and local search techniques.

3.1.4 *Mathematical Formulation of the Scheduling Problem*

The scheduling problem associated with the phase I of our 3-phase organizational design process is defined by the following parameters:

N = number of tasks to be processed.
K = number of available platforms.
S = number of resource requirement/capability types.
t_i = processing time of task i.
v_m = velocity of platform m.
$p_{ij} = \begin{cases} 0, \text{ if task } i \text{ must be completed before task } j \text{ can start} \\ 1, \text{ otherwise} \end{cases}$
r_{ml} = resource capability of type l on platform m.
R_{il} = resource requirement of type l for task i.
T = mission completion time found using a heuristic algorithm (or set to infinity).
0 = task that serves as "start-finish" (or "depot") task.

The following variables are used to define the scheduling problem:

Assignment variables:
$w_{im} = \begin{cases} 1, \text{ if platform } m \text{ is assigned to task } i \\ 0, \text{ otherwise} \end{cases}$
Traversing variables:
$x_{ijm} = \begin{cases} 1, \text{ if platform } m \text{ is assigned} \\ \quad \text{to process task } j \text{ after processing task } i \\ 0, \text{ otherwise} \end{cases}$
s_i = start time of task i.
Y = mission completion time (time when the last task is completed).

The problem constraints can be formulated as follows. Task i can be assigned to a platform m only if platform m travels to i directly from some other task j (including the depot task 0) and travels from this task i to some other task. The traveling of platform m is described by variables x_{ijm}. A platform can arrive at a task location (leave a task location) only once. Note that variables $x_{iim} = 0$ for $i=1,..,N$ (except for x_{00m} which can be 1 if the platform is idle during the entire mission). Therefore, the following constraints on the problem variables (called *assignment constraints*) are introduced:

$$\sum_{j=0}^{N} x_{ijm} = \sum_{j=0}^{N} x_{jim} = w_{im}, \ i = 0,...,N; m = 1,..,K$$

If task i must precede task j (that is, $p_{ij}=0$), then task j can begin to be processed only after task i is completed, that is

$$s_i + t_i \le s_j$$

This is true for all predecessors of task *j*. Also, if any platform *m* travels directly from task *i* to task *j* (that is, $x_{ijm}=1$), then task *j* can begin to be processed only after task *i* is completed plus the span of time needed for platform *m* to travel from *i* to *j* (this travel time is equal to d_{ij}/v_m), that is

$$s_i + t_i + \frac{d_{ij}}{v_m} \le s_j$$

Combining these together and noting that $T \geq s_i + t_i$ for any *i*, we obtain the following constraints (called *precedence constraints*):

$$s_i - s_j + x_{ijm} \cdot \left(\frac{d_{ij}}{v_m} + p_{ij} \cdot T \right) \le p_{ij} \cdot T - t_i$$

These constraints also eliminate cycling. When $p_{ij}=1$ and $x_{ijm}=0$, the precedence constraints are redundant.

Since the aggregated resource capability vector of a platform group assigned to a task should be greater than or equal to the task resource requirement vector, we obtain the following constraints (called *resource requirement constraints*):

$$\sum_{m=1}^{K} r_{ml} \cdot w_{im} \ge R_{il}, \qquad i = 1,..,N; l = 1,..,S;$$

These constraints also ensure that at least one platform is assigned to any task. The mission completion time is equal to the maximum among the completion times of all tasks. It is also not greater than the solution obtained by a heuristic algorithm. Therefore, the following constraints are introduced (called *mission completion time constraints*):

$$t_i + s_i \le Y \le T, \qquad i = 1,..,N;$$

The objective is to minimize the mission completion time. Then, the problem is formulated as follows:

$$\min Y$$

$$\begin{cases} \sum_{j=0}^{N} x_{ijm} - w_{im} = 0, \quad i = 0,...,N; m = 1,..,K; \\ \sum_{j=0}^{N} x_{jim} - w_{im} = 0, \quad i = 0,...,N; m = 1,..,K; \\ s_i - s_j + x_{ijm} \cdot \left(\frac{d_{ij}}{v_m} + a_{ij} \cdot T \right) \le a_{ij} \cdot T - t_i; i, j = 1,..,N; m = 1,..,K; \\ \sum_{m=1}^{K} r_{ml} \cdot w_{im} \ge R_{ml}, \quad i = 1,..,N; l = 1,..,S; \\ s_i - Y \le -t_i, \quad i = 1,..,N; \\ 0 \le Y \le T; s_i \ge 0; x_{ijm}, w_{im} \in \{0,1\} \end{cases}$$

This is a mixed-binary (i.e., containing continuous and binary variables) linear programming (MLP) problem (which is NP-hard). Moreover, even relaxing the constraints on the binary variables w_{im}, x_{ijm} (that is, making them real numbers in the [0,1] range) produces a linear programming problem (LP) with the number of variables equal to $K(N+1)^2 + N + 1$, the number of equality constraints equal to $2K(N+1)$ and the number of inequality constraints equal to $KN(N-1) + S(N+1)$. This creates "curse of dimensionality" and makes it hard to find solutions to even average-sized and relaxed scheduling problems.

3.1.5 *Optimal Solution via Dynamic Programming*

The optimal algorithms are based on the mixed-binary linear programming formulation described in the previous section. For more information on solving integer (binary) linear programming problems, see [Wosley, 1998], [Fang *et al.*, 1993], [Nemhauser, 1988], [Bertsekas, 1997]. The primary computational methods for solving mixed-integer programming problems optimally include the branch-and-bound algorithm, dynamic programming, column generation, and decomposition algorithms. The dynamic programming formulation for this problem is equivalent to the branch-and-bound algorithm with the following bounding rule: the n^{th} level of the branch-and-bound tree corresponds to the assignment of n tasks.

Define a state $(M, LT_1,..,LT_K, f_1,.., f_K)$, where $M \subset \{1,..,N\}$, LT_j is the task last processed by platform j, $LT_j \in \{0\} \cup M$, and f_j is its completion time. We associate with our problem a state space Φ of states $(M, LT_1,..,LT_K, f_1,.., f_K)$, where each state represents a feasible schedule of tasks from set M on platforms $1,\ldots,K$ such that the last task to be processed on platform j is LT_j, and it is completed at time f_j. The state space Φ can be decomposed as: $\Phi=\Phi_1\cup\ldots\cup\Phi_N$, where $\Phi_m = \{(M, LT_1,..,LT_K, f_1,.., f_K) \in \Phi, | M | = m\}$.

Then the solution to the scheduling problem is obtained as

$$Y = \min_{(M, LT_1,..,LT_K, f_1,.., f_K) \in \Phi_N} \max\{f_1,.., f_K\}$$

The states can be propagated from Φ_m to Φ_{m+1} in the following manner: for each state $(M, LT_1,..,LT_K, f_1,.., f_K) \in \Phi_m$, we can create a new state $(M', LT'_1,..,LT'_K, f'_1,.., f'_K) \in \Phi_{m+1}$ by assigning a task (such that it can be currently assigned – that is, all its predecessors are in *M*) to any of the group of platforms it can be assigned to. The information about the groups of platforms that can process each task can be either pre-computed off-line, or given as a problem parameter by the analyst. If task j is assigned to platforms $i_1,..,i_q$, then the new state $(M', LT'_1,..,LT'_K, f'_1,.., f'_K)$ has the following structure:

$$M' = M \cup \{j\}$$

$$LT'_i = \begin{cases} LT_i, \text{ if } i \notin [i_1,..,i_q] \\ j, \text{ if } i \in [i_1,..,i_q] \end{cases}$$

$$f'_i = \begin{cases} f_i, \text{ if } i \notin [i_1,..,i_q] \\ t_j + \max\left(\max\limits_{z \in IN(j)} f_j, \max\limits_{z \in [i_1,..,i_q]} \left(f_i + \dfrac{d_{LT_i j}}{v_z} \right) \right), \text{ otherwise} \end{cases}$$

The state space can be reduced by using the following two dominance and bounding tests.

Test 1: *Dominance*.
A state $(M, LT_1,..,LT_K, f_1,.., f_K)$ is said to dominate state $(M', LT'_1,..,LT'_K, f'_1,.., f'_K)$ if $f_j \le f'_j$ for each *j=1,..,K*. Clearly, the state $(M', LT'_1,..,LT'_K, f'_1,.., f'_K)$ can be discarded if such a state $(M, LT_1,..,LT_K, f_1,.., f_K)$ exists.

Test 2. *Bounding*.
Let $lb\{(M, LT_1,..,LT_K, f_1,.., f_K)\}$ denote a lower bound on the solution of the scheduling problem given that the assignments from state $(M, LT_1,..,LT_K, f_1,.., f_K)$ are fixed. It can also be considered as the solution to the scheduling problem with tasks $\{j : j \notin M\}$ and each platform *m* becoming available at time f_m. If this lower bound is greater than *T* (which is an upper bound on the optimal completion time), then this state can be discarded.

Example (continued).

In Figure 5, the state $(M, LT_1,..,LT_K, f_1,.., f_K) \in \Phi_9$ and its possible propagation is shown.

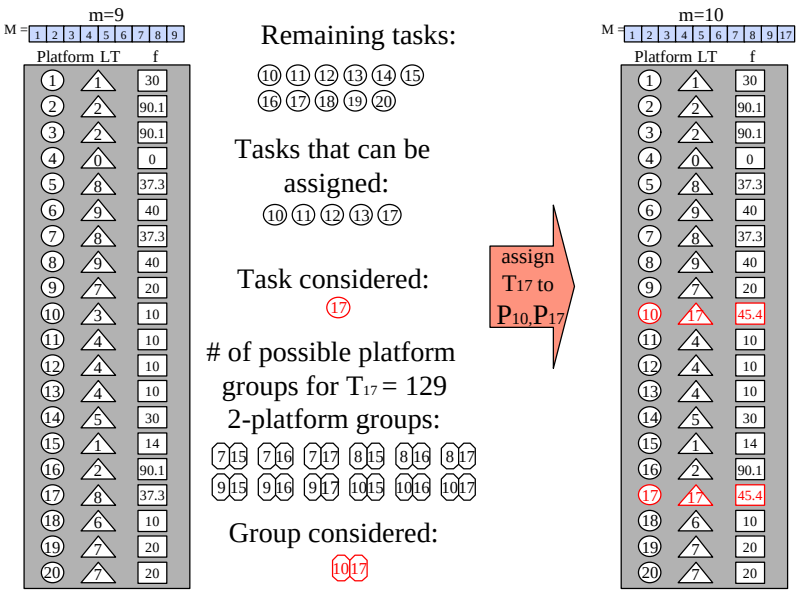

Figure 5. State Propagation for Scheduling Problem

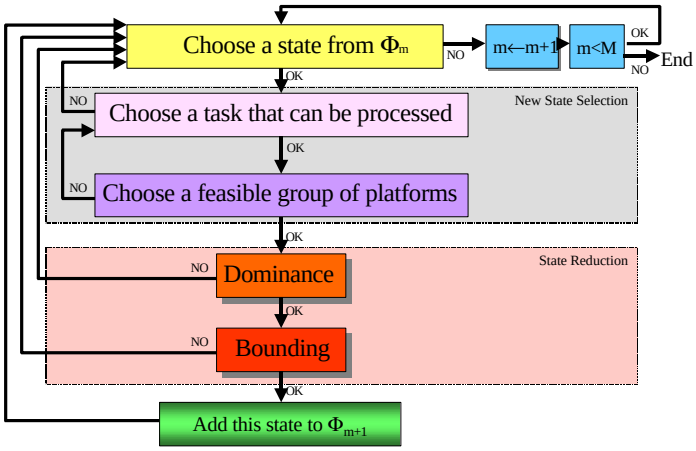

Figure 6. Dynamic Programming Algorithm

Different bounds can be used in Test 2. It was found that LP relaxation solution provides a close bound to the optimal (although the variables at which it is attained are not binary). In addition, relaxation techniques such as Lagrangian relaxation (described in [Levchuk *et al.*, 2000]) are used. Graphically, the dynamic programming algorithm is illustrated in Figure 6. This version of dynamic programming is equivalent to a breadth-first search in a branch-and-bound tree. It should be noted that dense precedence structures as well as tight lower and upper bounds substantially reduce the search space.

3.1.6 *Sub-optimal Algorithms*

3.1.6.1 *Dynamic List Scheduling Method*

The dynamic list scheduling (DLS) heuristic has two main Parts:

Part 1: Choose the task to be processed.
Part 2: Select the group of platforms to be assigned to it for processing.

The following notation (together with notations from section 3.1.1) are used throughout this section.

READY = set of tasks that can be processed at the current time
FREE = set of platforms available for processing tasks at the current time.
OUT(i) = set of direct successors of task i.
IN(i) = set of direct predecessors of task i.
nIn(i) = number of direct predecessors of task i.
nOut(i) = number of direct successors of task i.
CP(i) = critical path of task i (equal to the minimum required time from task i to the end of the mission).
level(i) = level of task i in the task precedence graph.
WL(i) = weighted length of task i.
$B(m,i)$ = amount of resources from platform m used to process task i.
$BR(m) = \sum_{i \in READY} B(m,i)$
$l(m)$ = last task processed on platform m (*0* if it has not processed any)
$G(i)$ = group of platforms selected for processing task i.
$FT=[f_1,..,f_M]$ –finish times of tasks that are currently being processed (assigned but not yet completed). Note that more than one task can have identical completion times.
$P(i)$ = priority coefficient assigned to task i in Part 1 of the algorithm.

The following three procedures were used for Part 1.

Critical Path Algorithm (CP). Critical paths *CP*(i) are calculated for each task given the task precedence graph and the task processing times. In the list scheduling algorithm, a task from *READY* is selected with the largest *CP*(i). When ties occur, task with the largest number of direct successors is chosen (or ties are broken arbitrarily). Priority values are set as $P(i) = CP(i)$.

Level Assignment Algorithm (LA). Levels are defined for each task based on the task precedence graph in a sequential manner. All predecessors of a task can be located only on lower levels (no task can have a direct successor in the same or lower level). The LA algorithm assigns tasks level by level. In the scheduling algorithm, a task from *READY* is chosen with the smallest level. When ties occur, task with the largest *CP*(i) is selected. Priority values are set as $P(i) = \max_j \{l(j)\} - l(i)$.

Weighted Length Algorithm (WL). As described in [Shirazi *et al.*, 1990], the following coefficient is used to select the tasks:

$$WL(i) = CP(i) + \max_{j \in OUT(i)} CP(j) + \frac{\sum_{j \in OUT(i)} CP(j)}{\max_{j \in OUT(i)} CP(j)}$$

While scheduling, task with the largest $WL(i)$ is selected. If ties occur, task with the largest $CP(i)$ is chosen (or ties are broken arbitrarily). Priority values are set as $P(i) = WL(i)$.

In Part 1 of the DLS algorithm, an assignment is considered whenever a task (or a group of tasks) is completed. At that time all the platforms processing the completed task become free (enter *FREE* set). All the tasks for which this task was the last processed predecessor become ready (enter *READY* set). Then, if there exists a task in *READY* set such that the aggregated capability vector of *FREE* set is component-wise more than or equal to this task requirement vector, an assignment can be made. Otherwise, the next completion time is considered.

In Part 2 of the DLS algorithm, we select a group of platforms to allocate to a task selected for processing in Part 1. The idea is to select platforms such that the amount of resources that are consumed by the task selected in Part 1 should affect the processing of other tasks in the *READY* set as little as possible. In addition, we want to choose the "closest" platforms in that the selected group of platforms can arrive at this task's location the fastest so as to minimize the completion time of the selected task. Each platform is assigned a coefficient and assignments are made in ascending order of these coefficients. The following coefficients were used (if task i is selected for processing in Part 1):

$$V_1(m) = \frac{B(m,i)}{BR(m) - B(m,i)}$$

$$V_2(m) = \left(s_{l(m)} + t_{l(m)} + \frac{d_{l(m),i}}{v_m} \right) \frac{B(m,i)}{BR(m) - B(m,i)}$$

$$V_3(m) = s_{l(m)} + t_{l(m)} + \frac{d_{l(m),i}}{v_m} + \frac{B(m,i)}{BR(m) - B(m,i)}$$

Here, s_j is the starting time of task j (with $s_0=0$ and $t_0=0$). After an initial group of platforms is found, it is then pruned by eliminating platforms from this group in descending order of these coefficients. The final group (which is irreducible) is allocated to task i and is denoted as $G(i)$.

When the platforms are assigned, the starting time for selected task i is computed as

$$s_i = \max\left(f, \max_{m \in G(i)} \left\{ s_{l(m)} + t_{l(m)} + \frac{d_{l(m),i}}{v_m} \right\} \right).$$

Parts 1 and 2 can be formalized as the following DLS algorithm:

DLS Algorithm.

Initialization. $READY=\{ i : nIn(i)=0 \}$, $FREE=\{1,..,K\}$, $M=0$.

STEP 1. Completion time Update. (skipped during initialization stage).

Pick $f = \min_{f_t \in FT}(f_t)$

$FT \neg FT \backslash \{f\}$

Let F_c be the corresponding group of tasks.

$FREE \leftarrow FREE \dot{E} G(F_c)$

for each $i \hat{I} F_c$

  for each $j \hat{I} OUT(i)$

    $nIn(j) \leftarrow nIn(j)-1;$

    if $nIn(j)=0$

      $READY \leftarrow READY \dot{E} \{j\}$

    end if

  end for

end for

STEP 2. Assignment Possibility Check.

if $\forall\, i \hat{I} READY\ \exists s : \sum_{m \in FREE} r_{ml} \le R_{il}$

  GO TO Step 1.

else GO TO Step 3

end if

STEP 3. Task Selection.

if $READY=Æ$

  GO TO Step 1.

end if

Find the set

$$READY1 = \left\{ i \in READY \,\middle|\, \sum_{m \in FREE} r_{ml} \ge R_{il}, l = 1,..,S \right\}$$

Select $i = \arg\min_{j \in READY1} \{P(j)\}$

$READY \neg READY \backslash \{i\}$

STEP 4. Platform Group Selection.

Find the set

$$FREE1 = \left\{ m \in FREE \,\middle|\, \sum_{l=1}^{S} \min(r_{ml}, R_{il}) \neq 0 \right\}$$

TG=∅

do until $\sum_{m \in TG} r_{ml} \geq R_{il}, \forall l = 1,..,S$

$n = \arg \max_{m \in FREE1} \{V_2(m)\}$

$FREE1 \leftarrow FREE1 \backslash \{m\}$

$TG \leftarrow TG \grave{E} \{n\}$

end do

STEP 5. Platform Group Pruning.

$n = \arg \min_{m \in TG} \{V_2(m)\}$

$TG1 = TG$

while $TG1 \neq \text{Æ}$

$n = \arg \min_{m \in TG1} \{V_2(m)\}$

$TG1 \leftarrow TG1 \backslash \{n\}$

if $\sum_{m \in TG \backslash \{n\}} r_{ml} \geq R_{il}, \forall l = 1,..,S$

$TG \leftarrow TG \backslash \{n\}$

end if

end while

STEP 6. Group Assignment.

$G(i) = TG$

$$s_i = \max\left(f, \max_{m \in G(i)} \left\{ s_{l(m)} + t_{l(m)} + \frac{d_{l(m),i}}{v_m} \right\} \right)$$

$f = s_i + t_i$

if $f \ddot{I} FT$

$FT \leftarrow FT \grave{E} \{f\}$

end if

GO TO Step 3.

Graphically, the process is illustrated in Figure 7.

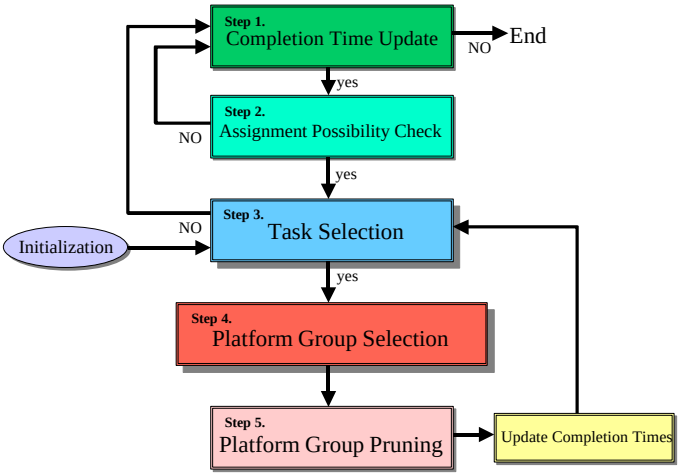

Figure 7. Dynamic List Scheduling Algorithm

Example (continued).

Consider the steps of DLS algorithm with the scheduling results shown as a Gantt chart in Fig. 8.

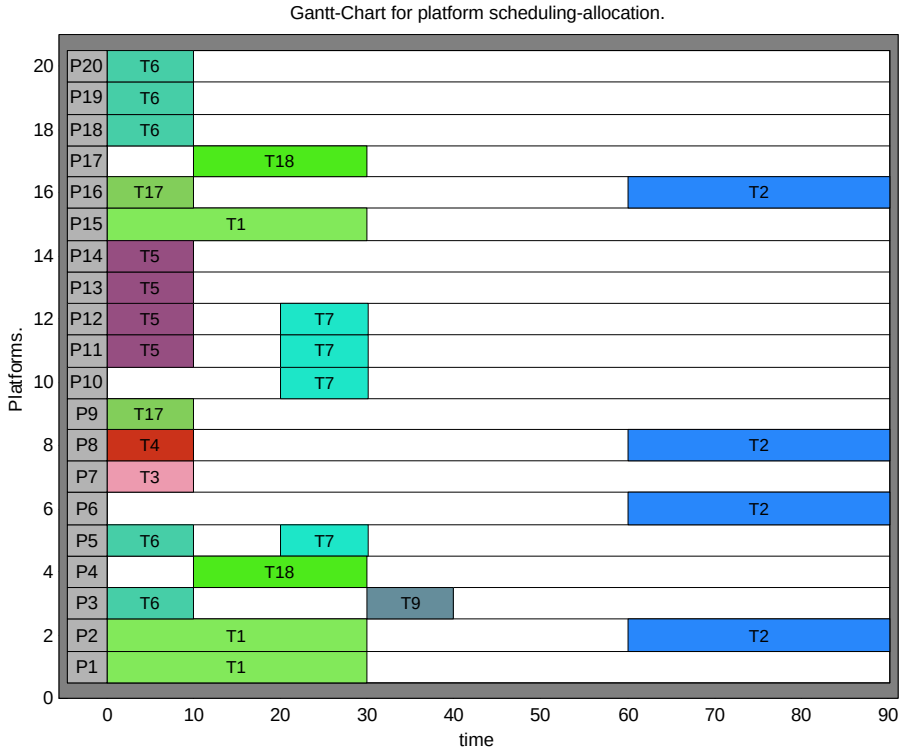

Figure 8. DLS step (task selection)

Note that a particular assignment for tasks *1* and *2* follows from the need to use platform *2* in processing both of them. The finish time of *30.0752 time units* is considered (this is a finish time of task *7*). Platforms *5,10,11,* and *12* are freed and the available platform set becomes *FREE*={*1,4,5,7,9,10,11,12,13,14,15,17,1,19,20*}. The set of *processed tasks* becomes {*1,3,4,5,6,7,17, 18*} and the set of *assigned tasks* becomes {*1,2,3,4,5, 6,7,9,17,18*}. Task *7* is the last processed predecessor of tasks *8* and *11*, so *READY*={*8,11*}. Both these tasks can be scheduled at this time by assigning them to platform groups {*9,11,13,18*} and {*7,14*} respectively. Moreover, task *8* is assigned first (because *CP*(*8*)=*55* and *CP*(*11*)=*25*). The groups are chosen according to the coefficients $V_2(\cdot)$ for platforms defined at this step. For task *8*, the corresponding platforms that can be assigned to it and their coefficients are shown in Figure 9.

The new schedule is shown in Figure 10 and the next completion time to be considered is *40 time units* corresponding to the completion time of task *9*. By completing task *9*, task *13* becomes available for processing. Note that, although the current mission processing time is *90.1*, we are "working inside" the mission.

The final scheduling results obtained by DLS are shown in Figure 11.

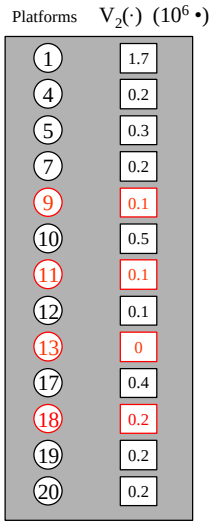

Figure 9. Platforms that can be used in processing task *8* and their preference coefficients.

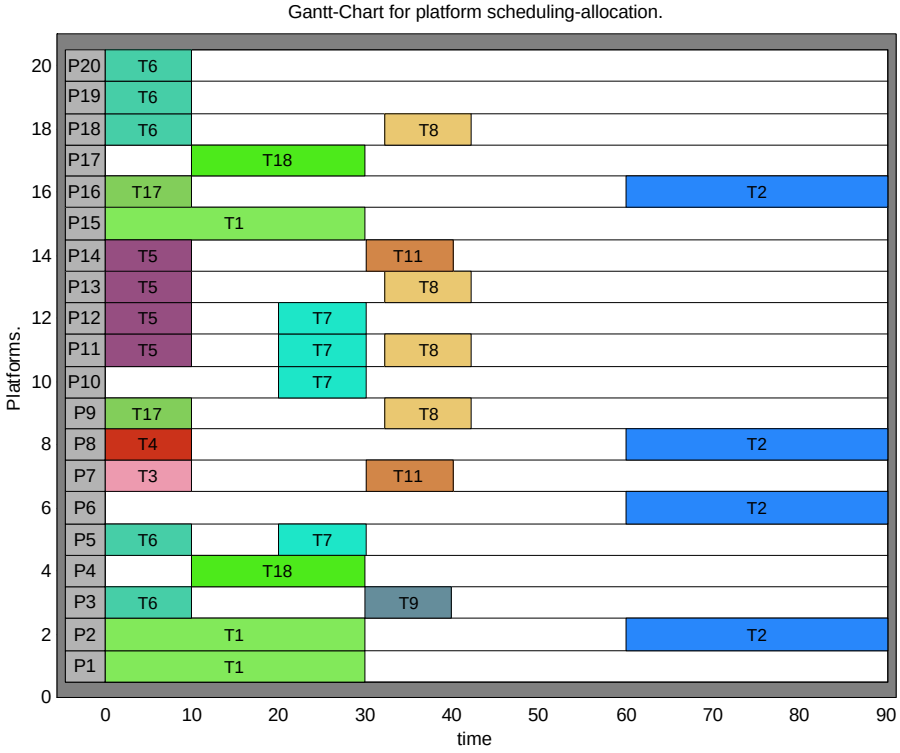

Figure 10. DLS step -current assignment completed.

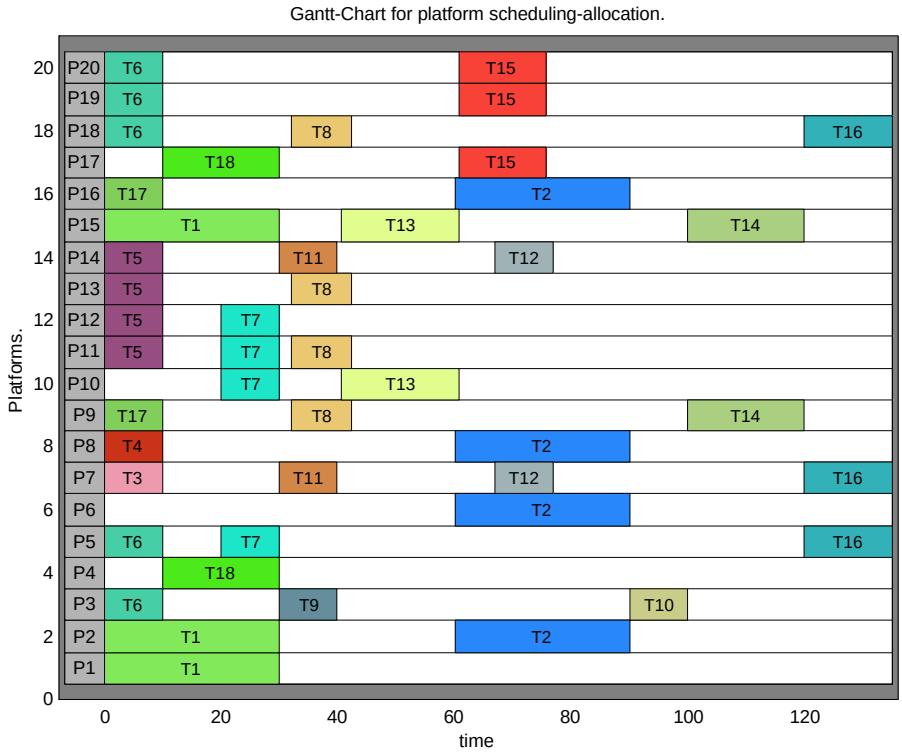

Figure 11. The Output of Scheduling Phase

3.1.6.2 **Pair-wise Exchange Improvement**

The DLS algorithm of subsection 3.1.5.1 produces sub-optimal solutions. It is expected that the sequence with which the tasks are assigned according to DLS is near-optimal. Suppose that the sequence of scheduling obtained from DLS is $i_1,..,i_N$. Then, the following algorithm is used to improve the scheduling results.

for *n=1:N-1*
do

Select $j \hat{I} \{n+1,..,N\}$ such that the scheduling sequence $i_1,...,i_{n-1},i_j,i_{n+1},...,i_{j-1},i_n,i_{j+1},...,i_N$ is feasible and the schedule obtained using platform allocation from DLS algorithm is the shortest one.

Then $i_1,...,i_N \leftarrow i_1,...,i_{n-1},i_j,i_{n+1},...,i_{j-1},i_n,i_{j+1},...,i_N$ (permute tasks i_n and i_j in the scheduling sequence).

end for

An exchange of tasks i_n and i_j is feasible ($n<j$) if

a) $IN(i_j) \subset \{i_1,...,i_{n-1}\}$

b) $OUT(i_n) \subset \{i_{j+1},...,i_N\}$

Typically, the pair-wise exchange heuristic produces 10 to 20% improvement in the completion time. However, in cases when DLS is already optimal or close to optimal, no improvement is obtained. In example 1, pair-wise exchange does not result in improvement. We can conclude that the output of DLS algorithm for example 1 is close to optimal.

Analyzing the scheduling results and resource requirement/capability data for example 1 proves that the schedule in Fig. 11 is optimal. It follows from the fact that tasks *1* and *2* have to be processed on platform *2*. Therefore, the fastest way to process these tasks is such that their finish times are 30 and 90.14 time units (since platform *2* must travel from one of them to another). When finish time of task *2* is 90.14, the earliest finish time for task *16* is 135.14 time units (which is equal to the mission completion time in Fig. 11). When finish time of task *1* is 90.14, then the earliest finish time for task *15* is equal to the mission completion time of Fig. 11, making it impossible to create a shorter schedule.

3.2 ***Resource Allocation to DM Nodes (Clustering)***

A cluster is comprised of a number of *similar* objects grouped together. In phase II of our organizational design process, the assignment results obtained in phase I are used to allocate platforms to decision-makers (DMs). A platform-task assignment gives us information about required coordination among platforms. This coordination among platforms stems from the need to process the same task; it is carried out through DMs assigned to these platforms as information/decision carriers. The coordination that occurs is one of information, decision, and action. Any two DMs are said to coordinate in processing a task if they are "owners" of platforms that are required simultaneously to process this task. Note that coordination of this sort can be avoided by assigning all the platforms (from a platform group assigned to process the task) to a single DM. In this case, many platforms may be assigned to a single DM. It results in increased coordination between the DM and the platforms processing a task. This is termed *internal coordination* of a DM. Conversely, the coordination among DMs is termed *external coordination*. This external coordination stems from two sources: direct one-to-one coordination among DMs and indirect coordination due to information flow through the DM hierarchy. In Phase II, only the workload due to direct one-to-one coordination among DMs is considered. Mathematical definitions of internal and external coordination are presented in subsection 3.2.1. For a review of clustering algorithms, see [Jain and Dubes, 1988].

Given the data from phase I, platforms are clustered into groups to be assigned to DMs. The objective is to minimize the DM coordination workload associated with DM-platform-task assignment. The workload is defined as a weighted sum of the internal and direct one-to-one external coordination, as well as the task workload.

3.2.1 ***Problem Formulation***

The following definitions are used to quantify the problem.

> *DM internal workload* is defined as the number of platforms assigned to a DM.
> *DM-to-DM direct external coordination* is equal to the number of identical tasks assigned to two DMs.

DM direct external workload is equal to the sum of DM's direct external one-to-one coordination with other DMs.

The following parameters are used to formulate the problem.

D = number of available DMs
B^I = bound on internal coordination workload allowed
B^E = bound on external coordination workload allowed
B^T = bound on number of tasks that can be assigned to a DM
W^I = weight on the internal workload
W^E = weight on the external workload
The platform-task assignment obtained in phase I is a matrix $[w_{im}]$ and is used as a parameter.

The following variables are used:

$$dp_{nm} = \begin{cases} 1, & \text{if DM } n \text{ is assigned platform } m \\ 0, & \text{otherwise} \end{cases}$$

$$ddt_{nmi} = \begin{cases} 1, & \text{if DMs } n \text{ and } m \text{ coordinate over task } i \\ 0, & \text{otherwise} \end{cases}$$

$$dt_{ni} = \begin{cases} 1, & \text{if DM } n \text{ is assigned to process task } i \\ 0, & \text{otherwise} \end{cases}$$

C_w = maximal weighted coordination workload

A DM n is assigned to a task i if and only if it is assigned to some platform which was assigned to process this task (this information was obtained in phase I). Therefore, the following constraints are introduced (called *DM assignment constraints*):

$$dt_{ni} \geq w_{im} \cdot dp_{nm} \quad \forall m = 1,..,K$$

Inequality in this formulation would become tight for some platform m. That is, we would have $dt_{ni} = \max_{m=1,..,K} w_{im} \cdot dp_{nm}$ after optimization (which is exactly the definition of variable dt_{ni}).

Two DMs n and m coordinate through task i if and only if they are assigned this task. It means that $ddt_{nmi} = \min(dt_{ni}, dt_{mi})$. Therefore, the following constraints are introduced (called *DM external coordination constraints*):

$$ddt_{nmi} \geq dt_{ni} + dt_{mi} - 1$$

The right-hand side is equal to 1 if and only if $dt_{ni} = dt_{mi} = 1$. Whenever this is not true, DMs n and m are not coordinating over task i and the variable $ddt_{nmi} = 0$ (and the right-hand side is ≤ 0).

The number of tasks assigned to a DM n is equal to $\sum_{i=1}^{N} dt_{ni}$. An internal DM workload is $\sum_{m=1}^{K} dp_{nm}$ and external DM workload is $\sum_{z=1, z\neq n}^{D} \sum_{i=1}^{N} ddt_{nzi}$. Therefore, the following constraints are

introduced (constraints on the number of tasks assigned to a DM, number of internal and external coordinations):

$$\sum_{i=1}^{N} dt_{ni} \le B^T$$
$$\sum_{m=1}^{K} dp_{nm} \le B^I$$
$$\sum_{z=1,z\neq n}^{D} \sum_{i=1}^{N} ddt_{nzi} \le B^E$$

The maximal weighted coordination workload is $\max_{\mathrm{n=1,..,D}} W^I \cdot \sum_{m=1}^{K} dp_{nm} + W^E \cdot \sum_{z=1,z\neq n}^{D} \sum_{i=1}^{N} ddt_{nzi}$.

Consequently, the constraints for maximal weighted coordination workload are

$$C_W \ge W^I \cdot \sum_{m=1}^{K} dp_{nm} + W^E \cdot \sum_{z=1,z\neq n}^{D} \sum_{i=1}^{N} ddt_{nzi}, \forall n = 1,..,D$$

The objective of Phase II is to minimize C_w. This results in a binary linear programming problem:

$$\min \ C_W$$
$$\begin{cases} dt_{ni} \ge w_{mi} \cdot dp_{nm}, \quad m = 1,..,K; n = 1,..,D; i = 1,..,N \\ ddt_{nmi} \ge dt_{ni} + dt_{mi} - 1, \quad m = 1,..,K; n = 1,..,D; i = 1,..,N \\ \sum_{i=1}^{N} dt_{ni} \le B^T, \quad n = 1,..,D \\ \sum_{m=1}^{K} dp_{nm} \le B^I, \quad n = 1,..,D \\ \sum_{z=1,z\neq n}^{D} \sum_{i=1}^{N} ddt_{nzi} \le B^E, \quad n = 1,..,D \\ C_W \ge W^I \cdot \sum_{m=1}^{K} dp_{nm} + W^E \cdot \sum_{z=1,z\neq n}^{D} \sum_{i=1}^{N} ddt_{nzi}, \quad n = 1,..,D \\ dt_{ni}, dp_{nm}, ddt_{nzi} \in \{0,1\} \end{cases}$$

Note that the variables [dp_{nm}] determine all the parameters (other variables and all the constraints) in the problem. This kind of problem structure makes it easier to apply optimal algorithms. Again, as in section 3.1.2, optimal algorithms such as dynamic programming and decomposition algorithms can be used to find the optimal solution.

3.2.2 ***Sub-optimal Algorithm: Hierarchical Clustering***

Assume that two DMs (*n* and *m*) are assigned the platform sets $\{n_1,...,n_U\}$ and $\{m_1,...,m_V\}$ respectively with the corresponding internal workloads *U* and *V*. We define the *assignment signature vector* for each such DM (group of platforms):

$$Q_n = [q_n, I_{n1},..., I_{nN}]$$

where q_n = number of platforms in the group (assigned to DM), I_{ni} = 1 if DM *n* is assigned task *i*.

Here, variables I_{ni} are determined as $I_{ni} = \max_{n \in DM} w_{in}$. Then, $U=q_n$, $V=q_m$ and external communication between DMs *n* and *m* is $\sum_{i=1}^{N} \max(I_{ni}, I_{mi})$.

Suppose that two platform groups $C_1 = \{n_1,...,n_U\}$ and $C_2 = \{m_1,...,m_V\}$ are to be combined into a new cluster at the next step of the algorithm. This would produce a decrease in external coordination for other DMs (because the coordination with one of them is eliminated). The decrease would be the most if vectors $[I_{n1},...,I_{nN}]$ and $[I_{m1},...,I_{mN}]$ were the same and equal to [*1*,…,*1*].

Clearly, we would want to combine the groups which are "close" under these conditions (carry close assignment signatures). Note that if these vectors have all distinct entries, then other DMs' external workloads would not decrease after these two groups are joined together. The "closeness" is defined as the number of 1's in the same places in the signature vectors $[I_{n1},...,I_{nN}]$ and $[I_{m1},...,I_{mN}]$. Also, when two groups are combined, the number of platforms (that is, internal workload) in the new group is the sum of the two old group sizes. We want to obtain a tradeoff between maximizing the "closeness" between the groups and minimizing the new group size. It is done by minimizing a weighted function using weights for internal and external coordination. The distance between two clusters $\{n_1,...,n_U\}$ and $\{m_1,...,m_V\}$ is then defined as

$$d(C_1, C_2) = d([q_n, I_{n1},..., I_{nN}], [q_m, I_{m1},..., I_{mN}]) = \\ = W^I(q_n + q_m) - W^E \sum_{i=1}^{N} \min\{I_{ni}, I_{mi}\} \quad (*)$$

The method of combining clusters in this way is called *hierarchal clustering*. The algorithm is as follows.

Step 1. Begin by assigning each platform to a distinct cluster. Define *assignment signature vector* for each cluster *m=1,..,K* as $Q_m = [1, w_{1m},..., w_{Nm}]$ (where w_{im} are platform-task assignment variables obtained in phase I). Define the distance between any two clusters as in (∗).

Step 2. Choose two clusters with minimum distance between them and combine them into a single cluster. Update the signature vectors and the distance matrix. If two clusters with signature vectors $Q_n = [q_n, I_{n1}, ..., I_{nN}]$ and $Q_m = [q_m, I_{m1}, ..., I_{mN}]$ are to be joined together, the new cluster with the following signature is obtained:

$$Q = [q_n + q_m, \max(I_{n1}, I_{m1}), ..., \max(I_{nN}, I_{mN})].$$

Step 3. If the number of clusters is equal to D (numbers of available DMs), the algorithm terminates.

Example (continued).

Given the results obtained in the scheduling phase, platforms are hierarchally clustered into $D=5$ clusters to be assigned to 5 available DMs. For internal and external coordination workload weights $W^I=1$ and $W^E=2$, the resulting coordination network is shown in figure 12. For workload weights $W^I=3$ and $W^E=2$, the resulting coordination network is a tree shown in Figure 13. DM-platform assignment and cluster signature vectors for these two examples are shown in Figure 14.

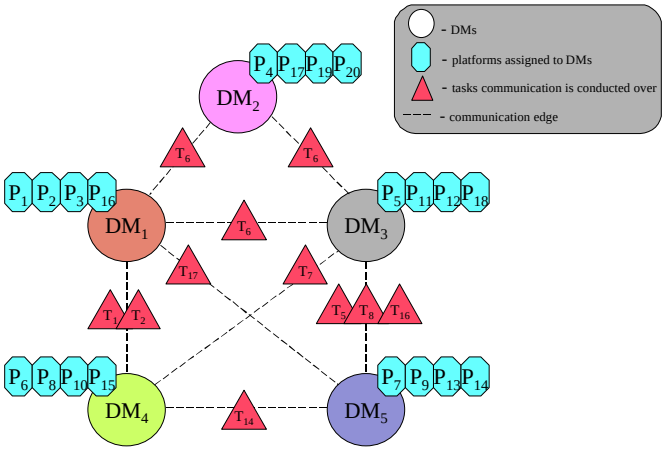

Figure 12. DM-platform allocation and Required Coordination for $W^I=1$, $W^E=2$

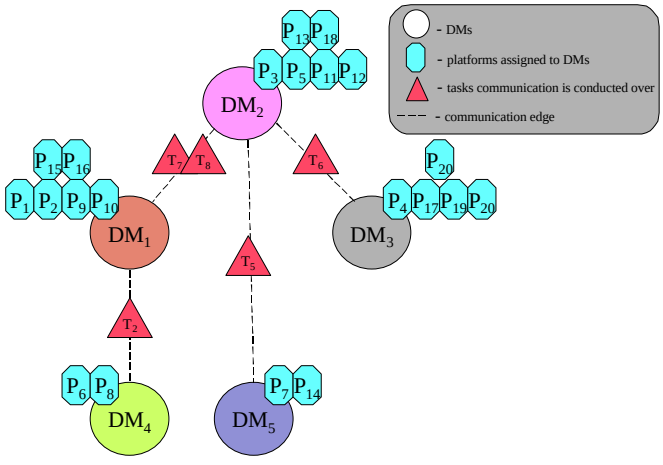

Figure 13. DM-platform allocation and Required Coordination for $W^I=3$, $W^E=2$

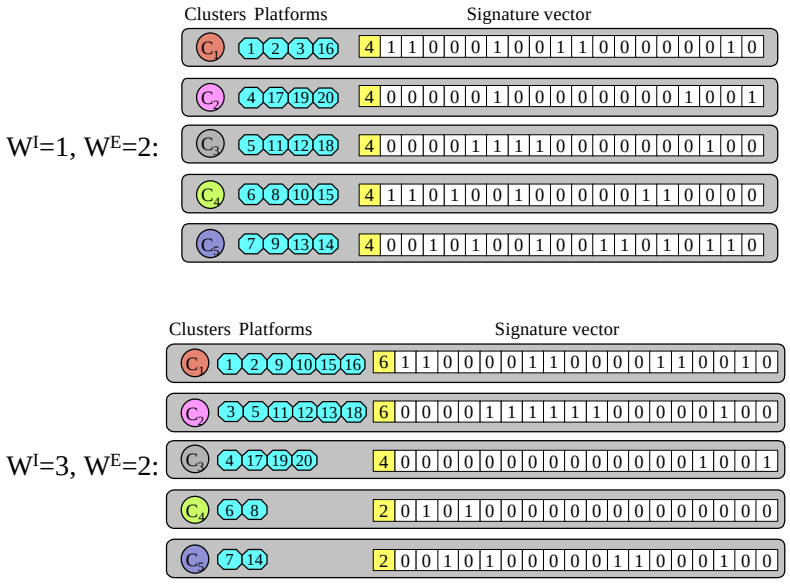

Figure 14. Platform clusters and their signature vectors

3.3 *Organizational Hierarchy*

In phase II, allocation of DMs to resources (platforms) is obtained. An external DM-DM coordination is determined based on joint task processing. DMs with their inter-DM coordination represent a network, where nodes are the DMs and edges denote coordination induced by joint task processing. Edge weights are equal to the required amount of coordination.

Hierarchical organizations eliminate decision-making confusion by imposing *superior-subordinate (supported-supporting) relations*. This means that organizations represent a layered structure where DMs from the lower level have exactly one link to the preceding level. The hierarchy consists of links through which it is permitted to communicate inside the hierarchy. These links form a tree in the network of DM nodes. The goal is to match the organizational hierarchy to the coordination network that is necessary for completing the mission. Different definitions of matching create different formulations of the hierarchy construction problem. When the necessary communication link between two DMs is not in the hierarchy, the information required for their coordination is passed through nodes on the path between them in the hierarchy. Such a path is unique for tree-structured systems. The communication value between these DMs is then added to each DM on the path between them as an additional workload. It is called *indirect coordination*. The *external coordination workload* is then the sum of direct (one-to-one) and indirect (through an intermediary) coordination. Evidently, this workload should be minimized in some sense. Here we present two problem formulations based on two main definitions of minimization: minimization of the maximal DM external coordination workload and minimization of the overall indirect (additional) coordination induced by the hierarchy.

3.3.1 *Min-Max Problem Formulation*

When the objective is to minimize the maximal external coordination workload, we impose additional constraints on the information flow. We restrict the indirect communication to go through only one intermediate DM. Information can be distorted while in transit, and additional intermediate nodes on the information path would increase the decision delay. Hence it is justified to consider restrictions, such as a single intermediate DM, to make organizations more responsive.

In the problem formulation, we introduce the dummy node *0* that would serve as a single-link root node. After the optimization is done, it is deleted from the tree while maintaining the tree structure.

The following variables are used to formulate the problem:

$$x_{ij} = \begin{cases} 1, & \text{if there is a direct link between } i \text{ and } j \text{ in the hierarchy} \\ 0, & \text{otherwise} \end{cases}$$

$$z_{ijk} = \begin{cases} 1, & \text{if } i \text{ and } j \text{ are connected through } k \\ 0, & \text{otherwise} \end{cases}$$

W_{MAX} = maximal DM hierarchy workload

The fact that we would use "direct" links accounts for the need to structure the hierarchy level by level. Then, direct links exist only from the higher level to the next lower level. In fact, the level structure of the hierarchy would be changed afterwards to place the DM with the smallest workload at the root of the tree.

The following parameters are used (with the outputs of phase II):

c_{mn} = required coordination between DMs n and m given by $c_{mn} = \sum_{i=1}^{N} ddt_{mni}$.

d_{mn} = 1 if DMs n and m must communicate given by $dd_{mn} = \max_{i=1,..,N} ddt_{mni}$

e_n =external workload (direct) given by $e_n = \sum_{m \neq n} c_{nm}$

i_n=internal workload of a DM n given by $i_n = \sum_{m=1}^{K} dp_{nm}$

The number of edges in the tree is equal to the numbers of nodes minus 1. Because of the fact that we have a dummy root-node, the number of nodes in our graph is *D+1*. The deletion of the dummy node should not disconnect the network. Since the structure on the nodes *1,..,D* should also be a tree, the following additional constraint is introduced:

$$\sum_{i,j=1}^{D} x_{ij} = D - 1$$

As mentioned earlier, a node at any level (except for the root) has a single connection to a node in the previous level. This means that there is only one link into each non-root node (for each i there exists only one j such that $x_{ji}=1$). Root node does not have any in-links. Therefore, the following constraints on the number of "in-edges" are introduced:

$$\sum_{j=0}^{D} x_{j0} = 0, \sum_{j=0}^{D} x_{ji} = 1, i = 1,..,D$$

If node i is at level l and there is a directed edge from i to j (that is, $x_{ij}=1$), then node j is at level $l+1$. Therefore, the following constraints are imposed:

$$l_j \geq l_i + 1 + (x_{ij} - 1)(D+1), \quad i, j = 0,..,D \text{ (note that } l_0 = 0)$$

Clearly, when $x_{ij}=1$, $l_j \geq l_i + 1$. Otherwise, $l_j \geq l_i - D$, which is always true (number of levels cannot be more than the number of edges, hence the right-hand side is ≤ 0). These constraints are also "non-cycling" implying that they impose a tree-structure on the organization.

If DMs i and j must coordinate, they either are connected directly, or through some other DM k. This connection is unique. Therefore, we obtain the following constraints:

$$x_{ij} + \sum_{k=1}^{D} z_{ijk} \geq dd_{ij}, \quad i, j = 1,..,D$$

Whenever $z_{ijk}=1$, then there are links between i and k and between j and k (in some direction). We have an edge between nodes i and k if and only if $x_{ik} + x_{ki} = 1$. Since level-constraints prohibit having more than two edges (in different directions) between any two nodes, we have the following relation between variables x_{ij} and z_{ijk}:

$$x_{ik} + x_{ki} + x_{jk} + x_{kj} \geq 2 z_{ijk}, \quad i, j, k = 1,.., D$$

The total external workload is found adding indirect external workload to the direct external coordination workload of phase II. Therefore,

$$W_{MAX} \geq i_n + \mathrm{e}_n + \sum_{i<j} z_{ijn} c_{ij}, \quad n = 1,..,D$$

The objective is to minimize W_{MAX}. Combining all constraints, our problem is a linear binary programming problem of the following form:

$$\min \; W_{MAX}$$

$$\begin{cases} \sum_{i,j=1}^{D} x_{ij} = D-1 \\ \sum_{j=0}^{D} x_{j0} = 0, \sum_{j=0}^{D} x_{ji} = 1, i = 1,..,D \\ l_j \geq l_i + 1 + (x_{ij} - 1)(D+1), \quad i, j = 0,..,D \\ x_{ij} + \sum_{k=1}^{D} z_{ijk} \geq dd_{ij}, \quad i, j = 1,..,D \\ x_{ik} + x_{ki} + x_{jk} + x_{kj} \geq 2z_{ijk}, \quad i, j, k = 1,..,D \\ W_{MAX} \geq i_n + \mathrm{e}_n + \sum_{i<j} z_{ijk} c_{ij}, \quad n = 1,..,D \\ x_{ij}, z_{ijk} \in \{0,1\} \end{cases}$$

When the solution to this problem is found, the "dummy" root node is discarded. Then the node with the smallest workload is found (workload of DM *n* being calculated as $i_n + e_n + \sum_{i<j} z_{ijn} c_{ij}$) and selected to be at the root of the organizational hierarchy. Other choices lead to different organizational structures. The levels are then updated accordingly.

3.3.2 ***Optimal Coordination Tree***

In this section, we present the optimal algorithm due to [Hu 82]. The objective is one of minimizing the additional coordination (indirect) introduced by the hierarchy. When two DMs *i* and *j* coordinate (their coordination equal to c_{ij}) and an edge (a communication link) between *i* and *j* exists in the hierarchy tree, coordination is direct and is added to each of the coordinating DMs. The overall coordination in this case would be $2^{\chi} c_{ij}$. When there is no direct link, coordination (indirect) is also added to all the DMs on the path between them (there will be # *edges-1* nodes on this path). Therefore, overall coordination is

$$c_{ij}^{\chi}(\#\text{ of edges between } i \text{ and } j \text{ in the hierarchy tree } + 1)$$

Hence, the overall external coordination for any hierarchy tree *T* is

$$COM(T) = \sum_{i=1}^{D} \sum_{j=i+1}^{D} c_{ij} \cdot (\#\text{ of edges between } i \text{ and } j \text{ in the tree } T + 1)$$

A tree *T* that minimizes the function *COM*(*T*) is called *Gomory-Hu tree* (also called *optimal coordination tree*). The following algorithm computes the Gomory-Hu tree (from [Hu 82]).

The following definitions are used:

original network –a graph of nodes (DMs) with edge weights c_{ij} (coordination between DMs *i* and *j*).
residual network –a network derived from the original network and current tree *T* and processed under the algorithmic steps; used in changing tree *T*.
clique –a set of one or more nodes of the original network; a node of the tree *T*.
condensing: a set of nodes is said to be condensed if it is combined into a single node called *aggregated node*. A weight of the edge between this new *aggregated node* and any other node *n* in the network is equal to the sum of edge weights in the original network between *n* and all nodes in this aggregated node; when two cliques are condensed, it is equivalent to condensing the set of original network nodes which constitute these cliques. That is, if two cliques $G_1=\{i_1,\ldots,i_k\}$ and $G_2=\{j_1,\ldots,j_m\}$ are to be *condensed*, the new node is $G=\{i_1,\ldots,i_k,j_1,\ldots,j_m\}$ and for any node *n* from the original network, the edge in the residual network is

$$c_{nG}^{new}=\sum_{v=1}^{k}c_{ni_v}+\sum_{u=1}^{m}c_{nj_u}$$

The new node is also a clique. The edge in the residual network between two cliques G_1 and G_2 is

$$c_{G_1G_2}=\sum_{v=1}^{k}\sum_{u=1}^{m}c_{i_v j_u}$$

expanding: a clique is expanded when all the nodes of the original network it consists of enter the residual graph as independent nodes.
minimum cut: in the network, minimum cut between two nodes *n* and *m* is defined as two sets $(X,\overline{X})$ such that $n\in X, m\in\overline{X}$ and the amount of flow between these sets, that is $\sum_{i\in X,\, j\in\overline{X}}c_{ij}$, is minimal.

The algorithm for finding the optimal coordination tree is as follows.

Initialization. Start with $|T|=1$, a tree *T* containing a single *clique* which consists of all nodes of the original network from Phase II.

Step 1. Select a clique *G* in *T* which consists of more than one node of the original network. Disconnect this clique in *T* (remove all edges incident to this clique in *T*). It breaks *T* into several connected components.

Step 2. Create a residual network by condensing each connected component into one clique (node) and expanding selected clique.

Step 3. Pick any two nodes *i* and *j* (original nodes) from the selected clique and find minimum cut $(X,\overline{X})$ in the residual network, $i\in X, j\in\overline{X}$.
Note: X (and $\overline{X}$) consist of condensed cliques of *T* and of nodes of the original network (from clique *G*).

Step 4. Create two new cliques G_1, G_2 in the tree *T* replacing selected clique with them:
$G_1=\{i\in G|i\in X\}, G_2=\{j\in G|j\in\overline{X}\}$.
Note that $G=G_1\cup G_2$. The following edges are created in *T* between these new cliques and other (old) cliques of *T*:
For each clique $N\,\hat{I}\,T$ connected to *G* in *T*:
a) if $N\,\hat{I}\,X$, then an edge between *N* and G_1 is created;
b) if $N\,\hat{I}\,\overline{X}$, then an edge between *N* and G_2 is created;
The edges are updated as described.

Step 5. If all cliques of *T* contain only single nodes of the original network, STOP.

Graphically, the algorithm is represented in Figure 15.

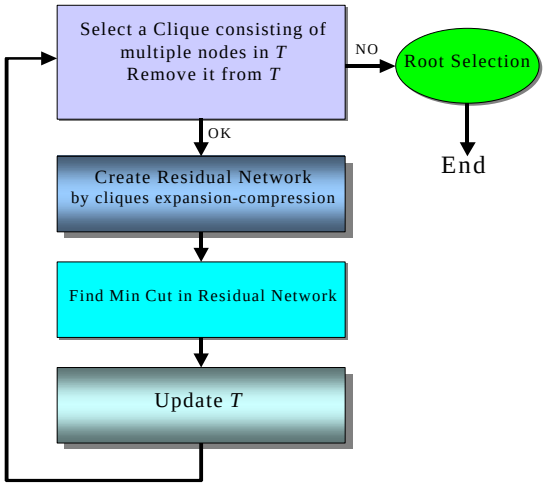

Figure 15. Optimal Coordination Tree Algorithm.

The complexity of the algorithm is polynomial in the number of nodes of the original network (number of DMs). In step 3, a min-cut algorithm (min cut=max flow) is used. Algorithms for min-cut problems include Ford-Fulkerson Algorithm (which can be exponential in the worst case but performs good in practice), DMKM, and other more sophisticated algorithms with polynomial complexity (see [Bertsekas, 1998]). When the tree is found, the node with the smallest overall workload is placed at the root of this tree.

Example (continued).

The network constructed from the coordination data obtained using workload parameters $W^I=1$, $W^E=2$ is given in Figure 16.

The step-by-step hierarchy structuring process is shown in the Figure 17. In the final step, the node with minimal workload is chosen to be at the root of the tree. The resulting organizational structure is shown in Figure 18. (Other choices will result in different organizational structures.)

Choosing the node with smallest weighted workload to be placed at the root of the tree is only one of the ways to structure the organization. Note that workload parameters $W^I=3$, $W^E=2$ produce tree-structured coordination network. Choosing the node with smallest workload to be a root results in a hierarchy depicted in Figure 18. On the other hand, choosing DM *3* would result in a hierarchy which can be viewed as more “responsive” because the commander (the root node) has closer access to other DMs. Selecting DM *2* to be a root results in an even better hierarchy with commander having direct access to all but one DM (Figure 21).

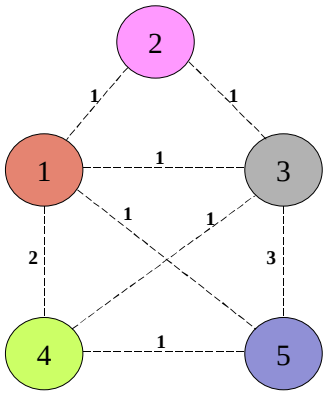

Figure 16. DM-coordination network, $W^I=1$, $W^E=2$

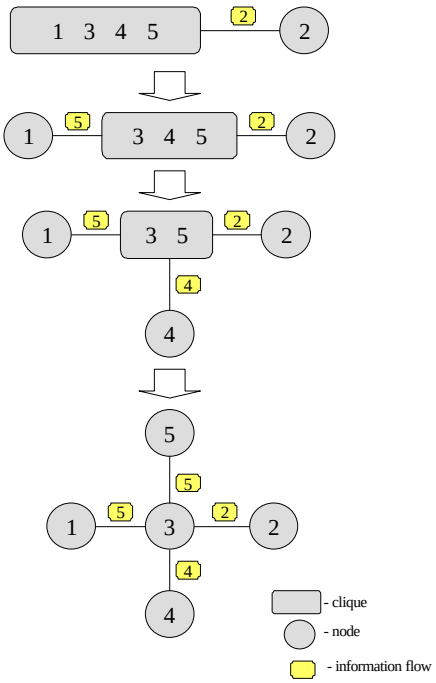

Figure 17. Hierarchy Construction

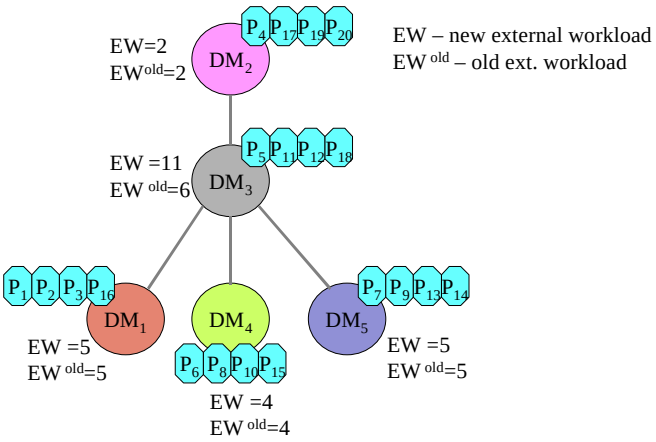

Figure 18. Organizational hierarchy for minimizing additional communication

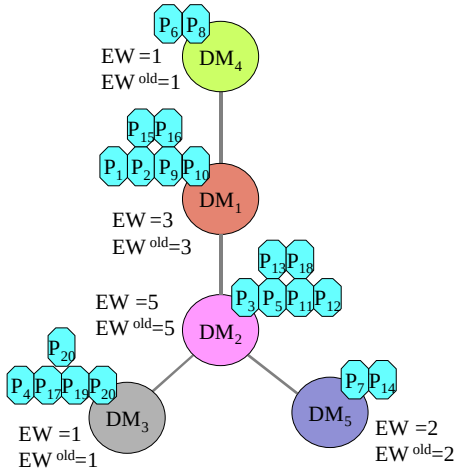

Figure 19. Organizational hierarchy for $W^I=3$, $W^E=2$ with *"min-root"*.

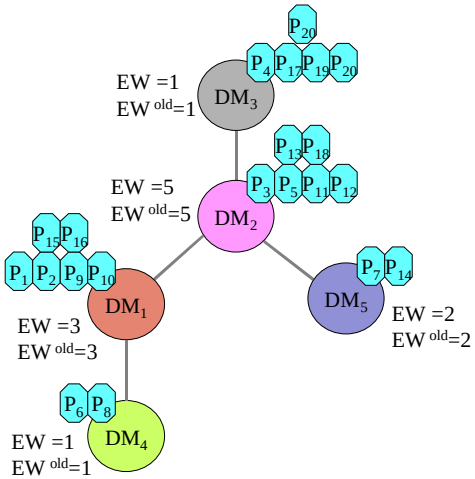

Figure 20. Organizational hierarchy for $W^I=3$, $W^E=2$ (with DM 3 as the root node).

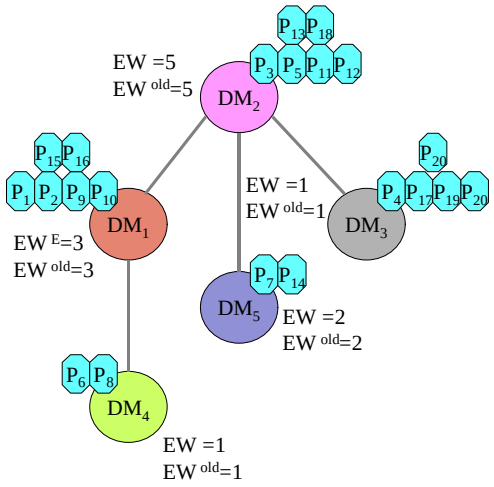

Figure 21. Organizational hierarchy for $W^I=3$, $W^E=2$ (with DM 2 as the root node).

3.3.3 *Maximal Spanning Tree Algorithm*

An alternative is to use maximal spanning tree algorithm to construct the organizational hierarchy tree. We obtain the tree T that maximizes $\sum_{(i,j)\in E(T)} c_{ij}$, where $E(T)$ denotes the set of edges of the tree T. This can be done by applying the minimum spanning tree algorithm. Note that the maximum spanning tree problem with edge weights c_{ij} transforms into a minimum spanning tree problem with edge weights $a_{ij}=c_{max}-c_{ij}$, where $c_{max}=max\{c_{ij}\}$. Methods for finding the minimal spanning tree include Kruskal, Jarnik-Prim-Dijkstra, and Bor'uvka (see [Bertsekas,98], [Hu,82]).

The algorithm is as follows:

Step 1. Select an edge with maximum coordination such that doesn't create cycles in the network.

Step 2. If ties occur, select the coordination link connected to the DM with minimal workload.

Step 3. When number of edges in the tree is equal to D (# of DM nodes), STOP.

The idea behind the algorithm is that we try to include the largest coordination links and to make DMs with largest workload to be at the lowest level of the hierarchy tree.

Example 1 (continued).

Constraining the depth of command to be at most 2, for the workload weights $W^I=1$, $W^E=2$, we obtain the tree shown in Figure 22.

4. Summary and Future Research

In this paper, we have presented the formulations and algorithms for three distinct phases of our organizational design process. Strict mathematical problem formulations provide the foundation

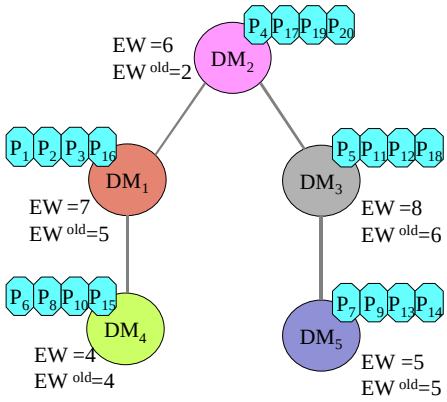

Figure 22. Organizational hierarchy for $W^I=3$, $W^E=2$ using maximal spanning tree.

for exploring ways to solve these problems with a required degree of optimality and choosing the specific algorithmic approaches according to available computational resources. Discussed problems are NP-hard, but their formulations allow one to introduce near-optimal polynomial algorithms.

Linear mixed-binary programming formulations allow one to construct approximation algorithms such as Lagrangian relaxation technique (creating a new problem by relaxing the constraints which are difficult to handle; for example, the resources constraints and precedence constraints in the scheduling problem formulation) and decomposition algorithms (decoupling the problem and solving simplified sub-problems, thereby reducing the size and computational complexity). Formulations of different Lagrangian relaxations and decompositions for scheduling, clustering and structural optimization phases of organizational design can be found in [Levchuk *et al.*, 2000]. These methods, together with mechanisms for adaptation, form the basis for our continuing research in this area.

Our current efforts are focuses on conducting a comparative analysis of various optimization algorithms in *solving specific design problems* and defining criteria for classifying multi-objective optimization problems into groups that require particular optimization sequence. This would allow us to *reduce solution complexity* for large-scale organizational design problems. Quantifying a set of user-defined *performance measures* provides the *criteria* for evaluating an organizational design. The above measures are aggregated to define an objective function for the design procedure. They also define measures of *organizational robustness* (i.e., the ability of an organization to maintain the required level of performance despite variations in its task environment) and of *adaptability* (i.e., the *ability of an organization to adapt* to environmental changes and functional failures). Developing fast algorithms for real-time analysis of feasible adaptation options, suggesting *suitable* forms of adaptation and appropriate *transition sequence* for reconfiguration would provide a computational framework for on-line adaptation in C2 systems.

5. **References**

[Baruah, 1998] S.K. Baruah. *The Multiprocessor Scheduling of Precedence-constrained Task Systems in the Presence of Interprocessor Communication Delays.* Operations Research, Vol. 46, No. 1, January-February, 1998, 65-72

[Barnhart *et al*., 1998] C. Bernhart *et al*. *Branch and Price: Column Generation for Solving Huge Integer programs.* Operations Research, Vol. 46, No. 3, May-June 1998, 316-329

[Bertsekas, 1998] D.P. Bertsekas. *Network Optimization: Continuous and Discrete Models.* 1998

[Bertsekas, 1997] D.P. Bertsekas and J.N. Tsitsklis. *Introduction to Linear Optimization.* 1997

[D. Bertsekas *et al*., 1992] D. Bertsekas et al. *Data Networks.* 1992

[Burns *et al.*, 1993] W.J. Burns and R.T. Clemen. *Covariance structure models and influence diagrams.* Manag. Sci,, vol. 39, 1993, 816-833

[Chan *et al*., 1998] L.M.A. Chan *et al*. *Parallel Machine Scheduling, Linear Programming, and Parameter List Scheduling Heuristics.* Operations Research, Vol. 46, No. 5, Sept-Oct 1998, 729-741

[Carley *et al*., 1995] K.M. Carley and Z. Lin. *Organizational Design Suited to High Performance Under Stress.* IEEE Transactions SMC, Vol. 25, 1995, 221-231

[Cheng *et al.*, 1990] T.C.E. Cheng and C.C.S. Sin. *A State-of-the-art Review of Parallel-Machine Scheduling Research.* European Journal of Operational Research, 47, 1990, 271-292

[Cheng *et al.*, 1994] T.C.E. Cheng and Z.-L. Chen. *Parallel-Machine Scheduling Problems with Earliness and Tardiness Penalties.* Journal of Operations Research Society, Vol. 45, No. 6, 1994, 685-695

[Current *et al*., 1994] J. Current *et al*. *Efficient Algorithms for Solving the Shortest Covering Path Problem.* Transportation Science, Vol. 28, No. 4, November 1994, 317-325

[Curry *et al.*, 1997] M.L. Curry, K.R. Pattipati, and D.L. Kleinman. *Mission Modeling as a Driver for the Design and Analysis of Organizations.* Proceedings of 1997 Command and Control Research and Technology Symposium, Monterey, CA, June 1997

[Diday *et al*., 1987] E. Diday *et al*. *Recent Developments in Clustering and Data Analysis.* Proceeding of the Japanese-French Scientific Seminar. March, 1987

[Dumas *et al*., 1995] Y. Dumas *et al*. *An Optimal Algorithm for the Traveling Salesman Problem with Time Windows.* Operations Research, Vol. 43, No. 2, March-April 1995, 367-371

[El-Rewini *et al*., 1994] H. El-Rewini *et al*. *Task Scheduling in Parallel and Distributed Systems.* 1994

[Franca, 1995] P.M. Franca. *The m-Traveling Salesman Problem with Minmax Objective.* Transportation Science, Vol. 29, No. 3, August 1995, 267-275

[Fischetti *et al*., 1997] M. Fischetti *et al*. *A Branch-and-cut Algorithm for the Symmetric Generalized Traveling Salesman Problem.* Operations Research, Vol. 45, No. 3, May-June 1997, 378-394

[Fisher *et al*., 1997] M.L. Fisher *et al*. *Vehicle Routing with Time windows: Two Optimization Algorithms.* Operations Research, Vol. 45, No. 3, May-June 1997, 488-492

[Fisher, 1994] M.L. Fisher. *Optimal solution of Vehicle Routing Problems using minimum K-trees.* Operations Research, Vol. 42, No. 4, July-August, 1994, 626-642

[Graham *et al.*, 1986] D. Graham and H.L.W. Nuttle. *A Comparison of Heuristics for a School Bus Scheduling Problem.* Transportation Research, Part B, Vol. 20, No. 2, 1986, 175-182

[Golden *et al.*, 1988] B.L. Golden and A.A. Assad. *Vehicle Routing: Methods and Studies.* 1988

[Hoffman *et al.*, 1993] K.L. Hoffman and M Padberg. *Solving Airline Crew Scheduling Problems by Branch-and Cut.* Management Science, Vol. 39, No. 6, June 1993, 657-682

[Hu, 1982] T.C. Hu. *Combinatorial Algorithms.* 1982

[Jain, Dubes, 1988] A.K. Jain and R.C. Dubes. *Algorithms for Clustering Data.* 1988

[Kempel, 1996] W.G. Kempel, D.L. Kleinman and M.C. Berigan. *A2C2 Experiment: Adaptation of the Joint Scenario and Formalization*. Proc. 1996 Command and Control Research and Technology Symposium, Monterey, CA, 1996.

[Kempel, 1996] W.G. Kempel, S.G. Hutchins, D.L. Kleinman, K. Sengupta, M.C. Berigan and N.A. Smith. *Early Experiences with Experimentation on Dynamic Organizational Structures*. Proc. 1996 Command and Control Research and Technology120 Symposium, Monterey, CA, 1996.

[Kempel, 1997] W.G. Kempel, J. Drake, D.L. Kleinman, E.E. Entin, and D. Serfaty. *Experimental Evaluation of Alternative and Adaptive Architectures in Command and Control.* Proceedings of the 1997 Command and Control Research and Technology Symposium, Washington, DC, June 1997.

[Kleinman *et al.*, 1996] D.L. Kleinman, P. Young, and G.S. Higgins. *The DDD-III: A Tool For Empirical research in Adaptive Organizations.* Proceedings of the 1996 Command and Control Research and Technology Symposium, Monterey, CA, June 1996.

[Lawler, 1976] E.L. Lawler. *Combinatorial Optimization: Network and Matroids.* 1976

[Lawler *et al*, 1985] E.L. Lawler *et al*. *The Traveling Salesman Problem.* 1985

[Levchuk *et al*., 1996] Y.N. Levchuk *et al*. *Design of Congruent Organizational Structures: Theory and Algorithms.* Proceedings of 1996 Command and Control Research and Technology Symposium, Monterey, CA, June 1996

[Levchuk *et al*., 1997] Y.N. Levchuk *et al*. *Normative Design of Organizations to Solve a Complex mission: Theory and Algorithms*. Proceedings of the 1997 Command and Control Research and Technology Symposium, Washington, DC, June 1997

[Levchuk *et al.*, 1998] Y.N. Levchuk, K.R. Pattipati , and D.L. Kleinman. *Designing Adaptive Organizations to Process a Complex Mission: Algorithms and Applications.* Proceedings of the 1998 Command & Control Research & Technology Symposium, NPS, Monterey, CA, June 1998.

[Levchuk *et al.*, 1999a] Y. Levchuck, K.R. Pattipati and D.L. Kleinman. *Analytic Model Driven Organizational Design and Experimentation in Adaptve Command and Control.* Systems Engineering, Vol. 2, No. 2 , 1999.

[Levchuk *et al.*, 1999b] Y.N. Levchuk, Jie Luo, Georgiy M. Levchuk, K.R. Pattipati, and D.L. Kleinman. *A Multi-Functional Software Environment for Modeling Complex Missions and Devising Adaptive Organizations.* Proceedings of the 1999 Command & Control Research & Technology Symposium, NPS, Newport, RI, June 1999.

[Levchuk *et al.*, 2000] G.M. Levchuk, Y.N. Levchuk, Jie Luo, Fang Tu, and K.R. Pattipati. *A Library of Optimization Algorithms for Organizational Design.* Dept. of ECE, Univ. of Connecticut, Cyberlab TR-00-102, Storrs, CT 06269-2157.

[Luenberger, 1984] D.G. Luenberger. *Linear and Nonlinear Programming.* 1984

[Madsen *et al*., 1995] O.B.G. Madsen *et al*. *A Heuristic Algorithm for a Dial-a-ride Problem with Time Windows, Multiple Capacities, and Multiple Objectives.* Annals of Operations Research, 60, 1995, 1993-208

[Malandraki *et al.*, 1992] C. Malandraki and M.S. Daskin. *Time Dependent Vehicle Routing Problems: Formulations, Properties, and Heuristic Algorithms.* Transportation Science, Vol. 26, No. 3, August 1992, 185-199

[Martello, Toth, 1990] S. Martello and P. Toth. *Knapsack Problems: Algorithms and Computer Implementations.* 1990

[Mingozzi *et al*., 1997] A. Mingozzi *et al*. *Dynamic Programming Strategies for the Traveling Salesman Problem with time window and Precedence Constraints.* Operations Research, Vol. 45, No. 3, May-June 1997, 365-377

[Nemhauser *et al*., 1988] G.L. Nemhauser and L.A. Wolsey. *Integer and Combinatorial Optimization.* 1988

[Papastavrou, 1992] J.D. Papastavrou and M. Athans. *On Optimal Distributed Detection Architectures in a Hypothesis Testing Environment.* IEEE Transactions on Automatic Control, Volume 37, 1992, 1154-1169.

[Perdu, Levis, 1997] D.M. Perdu and A.H. Levis. *A methodology for Adaptive Command and Control Teams Design and Evaluation.* Proceedings of the 1997 Command and Control Research and Technology Symposium, Washington, DC, June 1997.

[Pete *et al*., 1994] A. Pete, D.L. Kleinman, and K.R. Pattipati. *Structural congruence of tasks and organizations.* Proceedings of the 1994 Symp. on Command and Control Research and Decision Aids, NPS, Monterey, CA, 1994, 168-175

[Pete *et al.*, 1995] A. Pete, D.L. Kleinman, and K.R. Pattipati. *Designing organizations with congruent structures.* Proceedings of the 1995 Symposium on Command and Control Research and Technology, Washington, DC, 1995

[Pete *et al.*, 1996] A. Pete, K.R. Pattipati, and D.L. Kleinman. *Optimization of decision networks in structured task environments* IEEE Trans. Syst., Man, Cybern., Nov. 1996.

[Pete *et al.*, 1995] A. Pete, D.L. Kleinman, and P.W. Young. *Organizational performance of human teams in a structured task environment* Proceedings of the 1995 Symposium on Command and Control Research and Technology, Washington, DC, 1995.

[Pete *et al.*, 1998] A. Pete, K.R. Pattipati, D.L. Kleinman, and Y.N. Levchuk. *An Overview of Decision Networks and Organizations.* IEEE Trans. Syst., Man, Cybern. May. 1998, pp. 172-192.

[Pinedo, 1995] M. Pinedo. *Scheduling: Theory, Algorithms, and Systems.* 1995

[Reibman *et al.*, 1987] A. Reibman and L.W. Nolte. *Design and performance comparison of distributed detection networks.* IEEE Trans. Aerosp. and Electr. Syst., vol. 23, November, 1987, 789-79

[Shachter, 1986] R.D Shachter. *Evaluating influence diagrams.* Oper. Res., vol. 34, 1986, 871-882

[Shirazi *et al*., 1990] B. Shirazi *et al*. *Analysis and evaluation of Heuristic methods for static task scheduling.* J. of parallel and distributed computing 10, 1990, 222-232

[Shlapak *et al.*, 2000] Yurij Shlapak, Jie Luo, Georgiy M. Levchuk, Fang Tu, and Krishna R. Pattipati. *A Software Environment for the Design of Organizational Structures.* Proceedings of the 2000 Command & Control Research & Technology Symposium, NPS, Monterey, CA, June 2000.

[Tang *et al.*, 1993] Z.B. Tang, K.R. Pattipati, and D.L. Kleinman. *Optimization of Distributed Detection Networks: Part II. Generalized Tree Structures.* IEEE Transactions on Systems, Man and Cybernetics , vol. 23, 1993, 211-221.

[Turek et al., 1992] J. Turek, J. Wolf, K. Pattipati, and P. Yu. *Scheduling Parallelizable Tasks: Putting it all on the Shelf.* 1992 ACM Sigmetrics Conference, Newport, R.I., June1-5, 1992.

[Selvakumar *et al.*, 1994] S. Selvakumar and C.S.R. Murthy. *Scheduling Precedence Constrained Task Graphs with Non-Negligible Intertask Communication onto Multiprocessors.* IEEE Transactions on Parallel and Distributed Systems, Vol. 5, No. 3, March 1994, 328-336

[Solomon, 1987] M.M. Solomon. *Algorithms for the Vehicle Routing and Scheduling Problems with Time Window Constraints.* Operations Research, Vol. 35, No. 2, March-April 1987, 254-265

[Taillard *et al*., 1997] E. Taillard *et al*. *A Tabu Search Heuristic for the Vehicle Routing Problem with Soft Time Windows.* Transportation Science, Vol. 31, No. 2, May 1997, 170-186

[Van de Velde, 1993] S.L. Van de Velde. *Duality-Based Algorithms for Scheduling Unrelated Parallel Machines.* ORSA Journal on Computing, Vol. 5, No. 2, Spring 1993, 182-203

[Wolsey, 1998] L.A. Wolsey. *Integer Programming.* 1998

[Fang, 1993] S. Fang and S. Puthenpura. *Linear Optimization and Extensions. Theory and Algorithms.* 1993

[Wright, 1998] P.L. Wright and N.J. Asgford. *Transportation Engineering: Planning and Design.* 1998

[Xu, 1993] J. Xu. *Multiprocessor Scheduling of Processes with Release Times, Deadlines, Precedence, and Exclusion Relations.* IEEE Transactions on Software Engineering, Vol. 19, No. 2, Feb 1993, 139-154

[Ying *et al.*, 1997] Ying Jie, Y.N. Levchuk, M.L. Curry, K.R. Pattipati, and D. L. Kleinman. *Multi-Functional Flow Graphs: A New Approach to Mission Monitoring.* Proceedings of the 1997 Command and Control Research and Technology Symposium, Washington, DC, June 1997.

[Zweig, 1995] G. Zweig. *An Effective Tour Construction and Improvement Procedure for Traveling Salesman Problem.* Operations Research, Vol. 43, No. 6, November-December 1995, 1049-1057